DOCUMENT RESUME

ED 135 388 IR 004 499

AUTHOR	Kirby, Paul J.; Gardner, Edward M.
TITLE	Microcomputer Controlled, Interactive Testing Terminal Development.
INSTITUTION	Air Force Human Resources Lab., Lowry AFB, Colo. Technical Training Div.
SPONS AGENCY	Air Force Human Resources Lab., Brooks AFB, Texas.
REPORT NO	AFHRL-TR-76-66
PUB DATE	Oct 76
NOTE	27p.
EDRS PRICE	MF-$0.83 HC-$2.06 Plus Postage.
DESCRIPTORS	Autoinstructional Aids; Computer Programs; *Computer Science; *Individual Tests; Man Machine Systems; Self Pacing Machines; *Testing; *Test Scoring Machines
IDENTIFIERS	Microcomputers

ABSTRACT

The evolution of a self-contained test scoring terminal is presented. The rationale for the design is presented along with an evolutionary description of the requirements for the system. The sequence of software and hardware tools, which were developed in order to build the device, are also described in this report. The resulting device, which contains an imbedded microcomputer is functionally described and the testing strategies which it currently supports are presented. (Author)

****
* Documents acquired by ERIC include many informal unpublished *
* materials not available from other sources. ERIC makes every effort *
* to obtain the best copy available. Nevertheless, items of marginal *
* reproducibility are often encountered and this affects the quality *
* of the microfiche and hardcopy reproductions ERIC makes available *
* via the ERIC Document Reproduction Service (EDRS). EDRS is not *
* responsible for the quality of the original document. Reproductions *
* supplied by EDRS are the best that can be made from the original. *
****

AFHRL-TR-76-66

U S DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

AIR FORCE HUMAN RESOURCES LABORATORY

MICROCOMPUTER CONTROLLED, INTERACTIVE TESTING TERMINAL DEVELOPMENT

By

Paul J. Kirby, 1st Lt, USAF
Edward M. Gardner

TECHNICAL TRAINING DIVISION
Lowry Air Force Base, Colorado 80230

October 1976
Final Report for Period February 1975 – June 1976

Approved for public release; distribution unlimited.

AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235

ED135388

IR 004499

NOTICE

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related Government procurement operation, the Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This final report was submitted by Technical Training Division, Air Force Human Resources Laboratory, Lowry Air Force Base, Colorado 80230, under project 1121, with HQ Air Force Human Resources Laboratory (AFSC), Brooks Air Force Base, Texas 78235.

This report has been reviewed and cleared for open publication and/or public release by the appropriate Office of Information (OI) in accordance with AFR 190-17 and DoDD 5230.9. There is no objection to unlimited distribution of this report to the public at large, or by DDC to the National Technical Information Service (NTIS).

This technical report has been reviewed and is approved.

MARTY R. ROCKWAY, Technical Director
Technical Training Division

Approved for publication.

DAN D. FULGHAM, Colonel, USAF
Commander

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFHRL-TR-76-66
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): MICROCOMPUTER CONTROLLED, INTERACTIVE TESTING TERMINAL DEVELOPMENT

5. TYPE OF REPORT & PERIOD COVERED: Final, February 1975 – June 1976

6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): Paul J. Kirby, Edward M. Gardner

8. CONTRACT OR GRANT NUMBER(s):

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Technical Training Division, Air Force Human Resources Laboratory, Lowry Air Force Base, Colorado 80230

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62703F, 11210217 and 11210218

11. CONTROLLING OFFICE NAME AND ADDRESS: HQ Air Force Human Resources Laboratory (AFSC), Brooks Air Force Base, Texas 78235

12. REPORT DATE: October 1976

13. NUMBER OF PAGES: 24

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):

15. SECURITY CLASS. (of this report): Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report): Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):

18. SUPPLEMENTARY NOTES:

19. KEY WORDS (Continue on reverse side if necessary and identify by block number):
microcomputer
microcomputer assembly language
microcomputer hardware emulation
microcomputer software simulation
self-paced testing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number):

The evolution of a self-contained test scoring terminal is presented. The rationale for the design is presented along with an evolutionary description of the requirements for the system. The sequence of software and hardware tools, which were developed in order to build the device, are also described in this report. The resulting device, which contains an imbedded microcomputer is functionally described and the testing strategies which it currently supports are presented.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SUMMARY

Problem

The instructional environment created by self-pacing of mediated materials requires continual evaluation of student progress. This has been previously accomplished by explicit or embedded tests items presented to students who have recorded their responses on paper answer sheets. In the Advanced Instructional System (AIS), these sheets are currently read by a "management terminal" which is connected to a central computer that records student responses and makes appropriate instructional prescriptions. The operation of the reading device with marked sense forms has resulted in a very high rejection rate of forms, due to incorrect or unreadable information. The problem is to achieve the desirable aspects of this system while eliminating the many problems resulting from the use of paper forms.

Approach

The approach investigated under work units 1121-02-17 and 1121-02-18 has been to develop an electronic equivalent to the paper forms used in the current system. In this system the student responds to questions on a small device called a "microterminal" which consists externally of a keyboard and several display devices. Internally, this device contains an entire, miniature, programmable computer. This computer contains five different testing strategies and 900 random test answer patterns, and is capable of conducting an entire testing operation without intervention by any other computer. Provision is made for presenting the resulting data to a central computer through the AIS management terminal or to manually remove data from the device.

In order to produce this product, it was necessary to develop several software and hardware tools with which the microcomputer was simulated, programmed, and constructed. The development tools are described in this report along with the sequence of steps necessary to reach the final product which is the microterminal.

Results

The microterminal works as designed and offers a highly reliable alternative to the forms reader. The tools developed to build the microterminal are available to support modification of this device or construction of families of related devices.

Conclusions

The microterminal offers a solution to the problems of using the AIS management terminal and offers many further possibilities for low-cost, reliable testing systems within the Air Force.

PREFACE

We would like to acknowledge electronic technician Mr. Lyle McKnight's contribution for wiring and assembling both the hardware emulator and the microterminal, and for his continued assistance and support during the project. Mention should also be given to Mr. Joseph Lamos for some of his original conceptional ideas regarding a low-cost student terminal and for his role in the experimental testing of the device with 50 students from the Logistics Training School at Lowry AFB, Colorado, while using the first prototype of the present microterminal. Finally, mention should be given to LtCol Roger Grossel for providing managerial support and freedom in letting us pursue an idea which we believe could have significant impact on military training methodologies.

TABLE OF CONTENTS

		Page
I.	Introduction	5
II.	Product Evolution	5
III.	The Microcomputer Software/Hardware Development System	8
	The PIMPL Programming Language and Compiler	8
	Debugging the Initial Program – Software Simulation	11
	Hardware Development System	13
IV.	Description of the Student Microterminal	19
V.	Conclusion	21

LIST OF ILLUSTRATIONS

Figure		Page
1	Student microterminal	6
2	Student microterminal	9
3	Sample of PIMPL programming code	11
4	Hardware development system	14
5	Hardware development system	16

MICROCOMPUTER CONTROLLED, INTERACTIVE TESTING TERMINAL DEVELOPMENT

I. INTRODUCTION

The device described in this report is a prototype based upon experiences, both positive and negative, with the use of self-paced instructional materials in a military technical training classroom. The work was accomplished under two work units, 1121-02-17 and 1121-02-18. This report describes the evolution of a device and the technology which made it possible. It is the intent of this report to emphasize why the device has assumed its current characteristics, rather than to document its internal engineering in detail since these internal details could change significantly due to the incredible rate of change of microcomputer technology. However, since a special type of engineering environment is necessary for the development of devices of this kind, several sections of the report outline the tools which had to be built in order to develop the prototype hardware and software.

In order to accommodate a wider audience for this report, the product evolution is described in the first section, the development technology is described in the second section, and the resulting microterminal product in the third (Figure 1).

II. PRODUCT EVOLUTION

The concept of self-paced instructional materials is not new to either the military or civilian community. Such materials have been in use for a number of years in the form of programmed instructional texts, student operated media devices, and in limited applications in the form of computer assisted presentations at computer terminals. Due to the historically high cost of developing and validating instructional materials in all of these forms, the additional cost of the presentation media has often forced the actual materials to be presented in the cheapest of these forms, which is usually the printed format. As a corollary of this growth of paper materials, the need to manage students with widely differing learning rates has placed additional burdens upon the instructional staff. Because self-paced instruction accommodates these learning rates by altering student learning schedules, the group-paced, easy-to-manage conventional classroom has become the "learning center" with many different topics and techniques in use simultaneously by students proceeding at their own pace.

The attempts to automate management of the instructional process have been far more limited than the application of self-paced instructional materials. The most operational of these attempts has been at Memphis Naval Air Station, Tennessee, where self-paced materials have been presented with printed materials and student performance monitored with self-administered tests scored and recorded by computer. The use of the computer has enabled the frequent use of evaluation without imposing heavy burdens upon the instructional staff. A similar approach was the design basis for instructional management of the Advanced Instructional System (AIS) being developed for the United States Air Force at Lowry Air Force Base, Colorado.

The hardware and materials used in this concept consist of a "management terminal" and differing test answer forms which are filled in by students as instructional materials are executed. The terminal houses a forms reader, printing device and interface minicomputer in a carrel designed for student self-operation. In this instructional scenario, the student is expected to study materials, take a self-test, and present the resulting answer form to the management terminal. The terminal reads the form, transmits the information to the AIS central computer, and presents the test results and an instructional prescription to the student on the terminal printer. The central computer records the student's performance, produces periodic progress reports to his instructor, and retains information for instructional materials evaluation.

While this may sound like an ideal situation for a technical training classroom, there have been problems in practice, centering around the paper forms and reader. In order to provide response feedback to enhance learning, special chemically treated forms are used which darken upon the application of a special chemical crayon. It is intended that these marks be read by the reader and provide data for materials evaluation and for evaluation of the student's performance. In practice, the forms have proven difficult to read accurately, even when filled out properly by the students. This problem is accentuated by the fact that pencil marks used to identify the student and the test are spectrally different from the marks produced by the crayon.

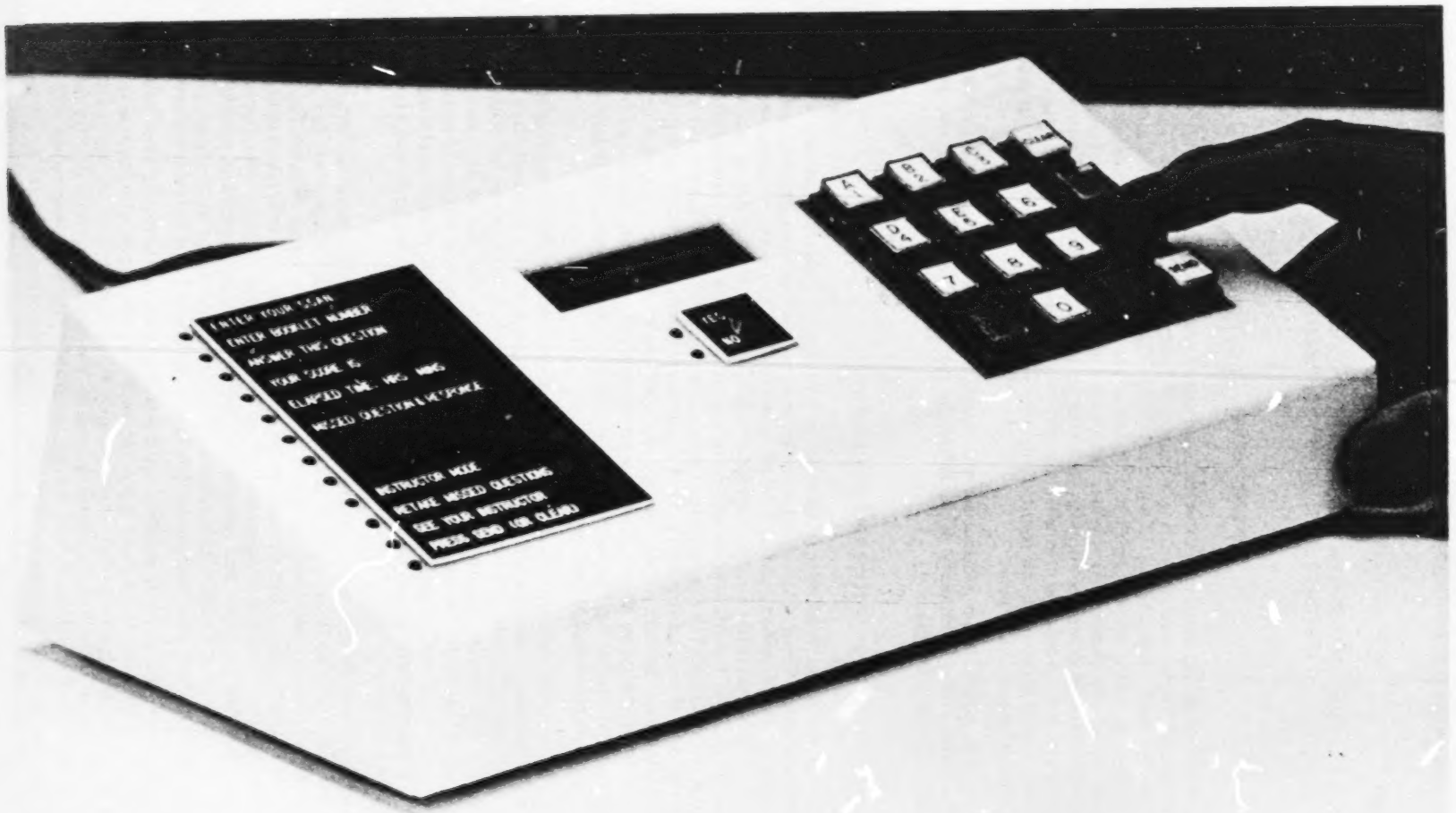

Figure 1. **Student microterminal.**

This problem led to the belief that an electronic approach to the problem might resolve these physical difficulties and possibly improve student performance by increasing the intensity of the student's interaction. The solution took the form of a device called a "student responder." In this device, a keyboard, several hexadecimal display elements (numbers 0..9 and letters A..F) and a column of individual display lamps were connected directly to a central computer through an interactive computer terminal. The device was driven by a program in the central computer which read the keyboard and illuminated the appropriate displays. This technique, while not satisfactory for a large number of responders, was adequate for experimental evaluation of the panel design and potential usefulness of the device.

To use the responder, the student would present his student I.D. number and his test booklet number to the device which would then tell him which items in the booklet to answer, and provide him with response feedback if appropriate. The program allowed several adaptive testing techniques and feedback modes.

In order to determine whether this device could function effectively with existing instructional materials, a programmed instructional text which was currently being used in an inventory management (supply) specialist course at Lowry was administered to a group of 50 students using the responder device. This text contains imbedded questions used to "exercise" the student and to determine whether he is mastering the material as he reads it. While no significant gains were noted in performance, the students were observed to cover the material about 30% faster on the average than students using chemically treated forms. In addition, 90% of the students favored the responder device over the chemical forms, suggesting that their acceptance of the device would pose no problem. All students questioned felt that the responder was easy to use and none required detailed personal instruction on its use.

Our initial plan was to produce an adequate number of these devices to allow for testing a larger group of students; in fact a printed circuit for the device was developed, along with an interface controller, to interface sixteen of the responders through a single access point to the central system. It was our intention to use a PDP-11 family minicomputer to perform the connection to the AIS terminal network hardware, primarily because this was the type of minicomputer used in the existing management terminal, but also because its manufacturers had announced inexpensive versions of this computer based on large scale integration electronics. Due to rapid advances in the technology of microprocessors, this approach was not followed. In the course of investigation of microprocessors to determine whether one could be used instead of the more costly PDP-11 to perform the interface task, we discovered that a vastly different approach to the responder was technologically cost feasible. We will now describe the steps taken in the design of the next version of the responder which (because of its stand-alone abilities) was renamed the mic oterminal.

Throughout the development of the original responder, a major concern was that the student would become dependent upon the central computer for many hours of continuous, errorfree hardware and software operation, since he would be using the responder intermittently for most of the instructional day. Thus, if the computer failed even once during, for example, a two-hour instructional module, the student would stand a good chance of being affected unless the central site were recording each minute interaction on disk; such an I/O usage at the central site would be detrimental if a large number of responders were employed. Also, the very infrequent requests encountered in individual responder usage would tie up more space at the central site than would be justified for the level of interaction which would be realized in a typical instructional situation. While this may seem paradoxical, it is typical of the instructional computer where peak to average processor requirements may vary by ratios larger than 1,000.

The solution to this problem, and prelude to many other possibilities, was to incorporate a complete microcomputer into the responder. Due to the large production potential of these devices, manufacturers had already begun to discount their prices to the point where a complete processor could be assembled for several hundred dollars in parts. The future promised decreases in these prices by at least a factor of four, thus suggesting that production prices of no more than the parts cost of the prototype might be realizable. It appeared that such a device could be deployed immediately since it would be unaffected by the reliability of the central site for its normal operation.

The first step in the design of the stand-alone terminal was to determine its design limits based not only upon its instructional requirements, but also upon the capabilities of existing technology to efficiently support these requirements. This is a delicate art of knowing where to draw the line between what is

wanted (which usually has no limits) and what can be cost-effectively achieved with the optimum combination of state-of-the-art components. In the time frame in which this work was completed, we benefited from the timely introduction of a series of microcomputer components by Motorola known as the M6800 family of parts. This enabled the construction of the microprocessor within a box which was nearly identical to the responder box in external dimensions. The external design of this box is indicated in Figures 1 and 2.

The remainder of this report is divided into several sections dealing with the characteristics of the responder and each of the tools which had to be developed to produce the prototype device. Because the microterminal is a self-contained computer, it must be programmed to perform the tasks associated with the identification of the student and the administration of the test or instructional material. This program had to be developed for a computer which did not yet exist; i.e., the one to be contained in the terminal. This "target" computer would have no facilities for the development of software since its normal function would be to score a test, so a more capable computer had to be obtained or built on which the software could be developed and tested. It is recommended that readers not interested in the details of this development process turn to Section IV.

III. THE MICROCOMPUTER SOFTWARE/HARDWARE DEVELOPMENT SYSTEM

The PIMPL Programming Language and Compiler

The specific microprocessor to be used in the microterminal was chosen for a number of reasons relating to cost, ease of use, ease of programming, and its ability to be constructed into a small physical container. At the initiation of this phase of the project, the Motorola M6800 microprocessor was the best choice considering all aspects of the design problem. Its weakest feature at the time, however, was that very little software existed to assist in the production of M6800 programs. Of the available software, all was incompatible with the development computers available to support the project. For this reason, we undertook the development of in-house software to aid in the programming of the microcomputer we were planning to build. Because of the background of the people involved in the project, it was decided that a large portion of the manpower involved would be spent in building good software tools, followed by a relatively short usage of these tools to program the prototype. Our reasoning was that the final product time would be about the same regardless of how much time was invested in the tools. By emphasizing tools rather than end product, we would be much better prepared for future modifications and developments in the microterminal or in related systems requiring a microprocessor for implementation.

A rather unique assembly language was developed, because the design constraints of size and cost mandated a highly compact program, while future needs suggested modifiability and versatility as important factors. The assembler program used to translate this language into M6800 machine code was written to run on the AIS CDC CYBER 73 central site computer, thus making it available to any Air Force installation having AIS terminals. The assembler is written in the PASCAL language which is excellently implemented on the central site computer. Because PASCAL is an ideal language for the construction of compilers and assemblers, the actual manpower required to write this program was about eight man-weeks. The acronym PIMPL stands for

P rogrammable
I nstructional
M icroprocessor
P rogramming
L anguage

A primary advantage in this implementation is that PIMPL programs may be edited using the powerful AIS program editor, submitted to the assembler for translation in a few seconds, and the resultant program transferred to the M6800 development processor for testing in a matter of minutes. Thus once these tools were available, the complete responder program was written in a matter of days and can be modified now in a matter of minutes.

The notion of an algorithmically formatted assembly language is attributable to Niklaus Wirth, who designed the assembly language PL360 for the IBM 360 series of full size computers. Previous experience with this language suggested that such an approach might be quite suitable for the programming of microcomputers. The language also bears some similarity in purpose to the PL/M language

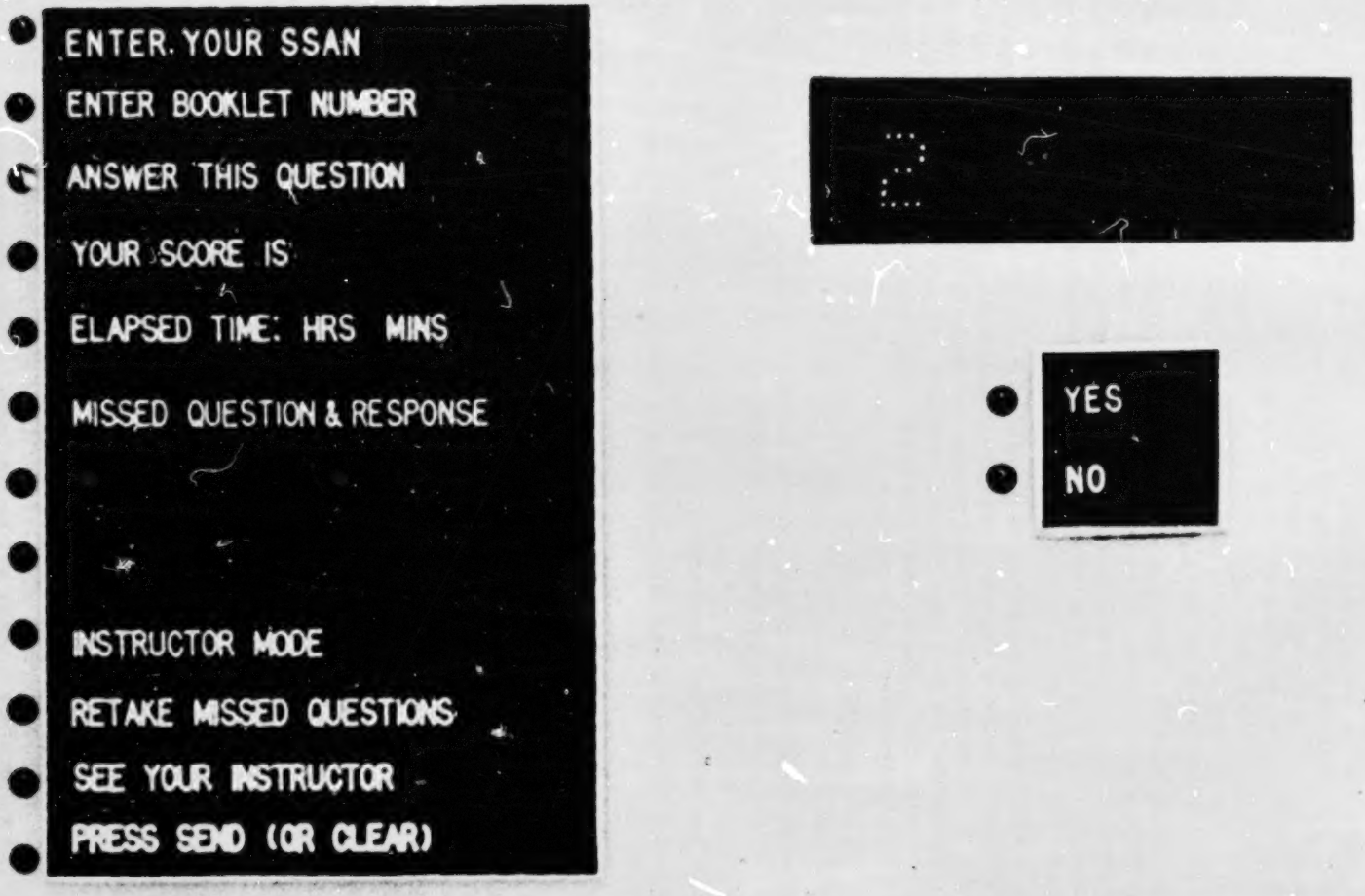

Figure 2. **Student microterminal.**

implemented for the Intel 8080 microcomputers, although it was intended that PIMPL would relate closely to the capabilities and limitations of the M6800 microprocessor. By including such language structures as procedures, IF-THEN-ELSE statements, and WHILE-DO statements such as found in high level programming languages, the flow of the resulting PIMPL program is much more transparent to the programmer. We feel that this significantly enhances the readability of the resulting program as well as reducing the number of logic errors in the program. Detailed statements, however, allow the efficiency which only most assemblers achieve and enable the resulting program to be small, thus reducing the cost, size, and power requirements of the finished product.

The assembler can be characterized as a top-down recursive descent translator, and appears to translate PIMPL at a rate of about 130 to 180 lines per second on the CYBER 73 computer. It requires approximately 40,000 octal words of machine storage for execution and is currently merged with the M6800 simulator program (described in the next section). This merger allows the immediate translation and simulation of PIMPL programs.

The PIMPL language allows the user to describe the storage of the target microcomputer with descriptive names of variables and constants used in the program. It uses attributes of these descriptions to check the syntax of the program as it is being compiled, and the feasibility of its execution in the target computer. For example, it will not allow storage allocated in a "read only" type of memory to be used in such a way that its value would be assumedly changed as the program runs. The bulk of executable statements are the assignment or function type, such as

```
A ← 5+B AND BITMASK          or
CALL TIMER
```

These are translated directly into machine code for the M6800 microprocessor. When an iterative condition is needed to express an idea requiring repeated test or looping, a statement such as

```
WHILE A<5 DO
BEGIN LIGHT← −LIGHT; A←A+1 END:
```

might be used to express the concept. If the user wishes to make a conditional execution he might use a form such as

```
IF A=0 THEN B←5 ELSE CALL SOMEFUNCTION      or
IF A<B THEN BEGIN A←B; B←0 END .
```

Statements may be grouped together into procedures such as

```
SOMEFUNCTION: PROCEDURE;
A←1; B←5;
WHILE A<B DO
  BEGIN A←A+2; B←B+1 END;
RETURN;
```

which may be called by statements such as

```
CALL SOMEFUNCTION;
```

In the event that an error is detected in the program by the compiler, it produces an error message of the form:

```
X ← X+1  CALL TIMER;
- - - - - - - - - - SEMICOLON EXPECTED
```

notifying the programmer where the error was detected, and, if possible, what caused the error. In this way the time consumed correcting typographical or feasibility errors is minimized.

By maintaining a policy of only implementing capabilities which are directly supported by the M6800 microprocessor, we kept the compiler small, fast, and easy to write and debug. We feel that the resulting small size of the responder program in relation to what it does verifies the correctness of this approach for the M6800. A section of the responder program is included in Figure 3 to demonstrate how well the language can describe the flow of the microprogram when properly used. The example was the first program written by one of the authors, who is not a professional programmer. Without being familiar with

```
[THIS ROUTINE GENERATES FRACTION OF A SECOND CLOCKS FROM
 A ONE MILLISECOND HARDWARE INTERRUPT CLOCK]

CLOCKINTERRUPT:PROCEDURE;
        IF DELAY>0 THEN DELAY:=DELAY-1:[FOR CALLED DELAY]
        MSCOUNT:=MSCOUNT-1:  [FOR SOFTWARE CLOCK]
        IF MSCOUNT=0 THEN
          BEGIN  [BLOCK DONE ONLY EVERY 10TH SEC]
            A:=10: MSCOUNT:=A:  [RESET 10TH SEC CTER]
            TENTHCOUNT:=TENTHCOUNT-1:
            A:=99: IF A<>HOURS THEN
            IF TESTFLAG<>0 THEN
            IF TENTHCOUNT=0 THEN
              BEGIN  [BLOCK DONE ONCE PER SEC]
                A:=10: TENTHCOUNT:=A:
                SECONDS:=SECONDS+1: A:=SECONDS:
                IF A=60 THEN
                  BEGIN
                    SECONDS:=0: MINUTES:=MINUTES+1:
                  END:
                A:=MINUTES:
                IF A=60 THEN
                  BEGIN
                    MINUTES:=0: HOURS:=HOURS+1:
                  END:
              END:
          END:
        A:=PA: [RESETS IRQA FLAG IN PIA]
        RETURN:
```

Figure 3. **Sample of PIMPL programming code.**

anything about the program, you could probably determine what this routine does from the information in the listing. This desirable trait is called "self-documentation" and is not commonly found in assembly language programs.

When the compiler has finished translating the program into M6800 machine instructions, it records these instructions in the disk system of the CYBER computer for later transfer to the microcomputer at the request of the programmer. It then prints a readable listing of what the contents of the target computer will be, so that this information may be used while debugging the program or for purposes of ordering read only memory circuits if desired. The microcomputer can also store this information in field programmable read only memories if desired. Normally, at this point, control will be transferred to the simulator program for trial execution.

Debugging the Initial Program – Software Simulation

When a PIMPL program has been successfully compiled into machine language, it may still contain logic errors which would not be revealed until it begins to execute. At the normal time of execution, however, the program will exist in a miniature, minimally configured box designed to perform a specific

limited function, which does not include the development and debugging of computer programs. In order to debug the program, all of the features of large scale, high-speed computers would prove beneficial. To obtain these benefits, the microcomputer to be built is "simulated" by a computer program running on the AIS CYBER computer. This simulator program can perform monitoring and checking tasks which would be infeasible to build into hardware devices specially designed to debug microprograms.

In order to minimize the user time required by the simulation step, the simulator program should contain very prolific information which is not obtainable in the high-speed execution environment of the actual hardware, but which is useful in determining whether the program is operating correctly. The simulator written for this project includes the following features:

1. control of maximum simulation run time
2. full display of microprocessing unit (MPU) internal registers
3. assignment to program counter
4. simulation traces initiated by accessing within a range of addresses
5. simulation traces initiated by MPU clock times
6. simulation traces controlled by opcodes executed
7. timed interrupts to write or alter data in memory
8. timed external MPU hardware interrupts
9. timed controls for partial memory and variable listings during simulation.

During simulation, a trace is printed for each machine instruction. Each line of trace lists the line number, program counter, index register, stack pointer, A register, B register, six condition code flags, cumulative MPU clock time, and new program counter. By correlating the trace listing with the program listing, it is quickly possible to analyze and debug logical errors in the source code.

Each simulation feature is controlled by a one-line data card attached at the end of the PIMPL source program. The first number of the line selects the type of simulation command (i.e., one of the features listed above). The remaining one to four numbers give the parameters of that command, such as designating at what time the interrupt should occur, or identifying at what program address the simulation listing should begin. Following is an explanation of each type of control card.

Simulation Run Time: command designator number 0 followed by one number for the maximum desired run time, expressed in microseconds. For example, a simulation data card (i.e., one line of data attached to the end of the PIMPL program) with 0 2000 will permit the simulation to run for 2,000 cumulative MPU (microprocessing unit) microseconds before terminating the simulation run.

Initialization of Program Counter: command designator number 1 followed by one number for the decimal value of the new desired program counter. If this command is not used, the program counter is established by the normal hardware convention of using the microprocessor startup vector which points to the starting address of the program.

Simulation Trace by Address: command designator number 2 followed by two numbers for the beginning and ending program counter addresses for tracing and listing the simulation. Any program code executed between these two addresses will be shown in the simulation trace. For example, a control data line with 2 64000 64200 will provide a simulation trace whenever that part of the program being executed is between program addresses 64,000 and 64,200.

Simulation Trace by Clock: command designator number 3 followed by two numbers for the beginning and ending cumulative MPU time for which the simulation trace should be listed. For instance, a data line with 3 2000 5000 would allow a simulation trace to be listed when the MPU is running between 2,000 and 5,000 microseconds of the total simulated real-time run.

Simulation Trace by Opcode: command designator number 4 followed by one number for the opcode. Whenever the microprocessor instruction with that opcode is executed, a one-line simulation trace is provided. As with all the simulation commands, more than one may be used by having multiple data lines attached at the end of the PIMPL program deck.

Timed Interrupts: command designator number 6 followed by four numbers. There are four types of interrupts used, a hardware interrupt of the MPU, a nonmaskable hardware interrupt of the MPU, an interrupt that will write or alter a byte of data in memory, and an interrupt that will list a designated page of memory at a predetermined time. This last type is transparent to the actual simulated program, but provides the user with a current list of memory and stored source program variables. Each of these interrupts is designated by the fourth parameter number, numbered 0 to 3.

The first of the four numbers designates at what time the interrupt is to occur, expressed in microseconds accumulated from the start of the run. The second number tells the memory address location. The third number is the data that is to be written into the memory address of the second number when the fourth number is a 2, which specifies a write into memory. And the fourth number, 0 to 3 in value, designates the type of interrupt.

An example of a hardware interrupt would be 6 2000 0 0 0. Here the first number, 6, indicates an interrupt. The number 2,000 indicates the interrupt is to occur at a time of 2,000 microseconds into the simulation. The next two zeros are ignored while the last 0 indicates this is to be a hardware interrupt as opposed to a nonmaskable, write memory, or page dump of memory interrupt.

An example of a nonmaskable interrupt would be the same except for the last number, 6 2000 0 0 1. Here the 1 distinguishes a nonmaskable interrupt from a normal interrupt.

An example of writing data into memory at a given time would be 6 2000 65000 77 2. This is interrupted by the simulator as, "at a cumulative MPU time of 2,000 microseconds write into memory location 65,000 the value 77."

The fourth type of interrupt is the page dump. Here 6 2000 255 0 3 is interrupted by the simulator to mean, "at a cumulative MPU run time of 2,000 microseconds, print the contents of the 256th page (pages numbered from zero) of memory and then continue with the simulation." This is a transparent interrupt with the result that all memory locations between hexadecimal FF00 and FFFF will be printed at that point of time in the simulation listing.

As mentioned, each type of command card may be used many times. As an example, suppose a program is quite extensive and involves a large amount of simulation time. Simulation traces controlled by address, time, or opcode may be discriminately turned on and off, listing only those portions of the simulation the author is concerned with. A simulation representing a microprocessor controlled sequence of events or interactions with the external world could conceivably take several real-time seconds or minutes. The program trace can easily be configured to follow only those interactions the author is presently concerned with. Software delays and wait loops may be skipped and counters may be changed–jumping to new segments of executable code. Interrupts may be interjected to simulate external interactions with switches, controls and signals.

Upon completion of the simulation, a listing is provided of all declared random access memory (RAM), read only memory (ROM) and I/O (input/output) ports. This is given by page number, byte number, decimal and hexadecimal value, and designates whether it is RAM, ROM, or an I/O port, and loaded or unused space. This type of listing is useful for a number of reasons. It shows the amount of object code generated by the assembly, the final value of all the program variables stored in RAM, the location in memory of specific procedures, the actual object code generated by the assembler, an overall mapping of memory and I/O ports which is useful for correlation with actual hardware addressing schemes, and a means of verifying object code placed in ROMs or PROMs (programmable ROMs).

Hardware Development System

Once a good language, assembler and simulator have been established, the final component necessary for a complete microcomputer development system is a hardware emulator (Figure 4). It should be a working, self-contained microcomputer that not only duplicates the microcomputer (proposed in the final hardware design) but also includes many features to facilitate program development and debugging.

It may initially take longer to develop such a system, but once completed it will pay for itself many times in time saved during the microcomputer development process. The capability to "look inside" the microcomputing process, halt and display information, read or write memory data, and continue running becomes an invaluable aid. With this tool, if properly designed and used, development efficiency increases significantly.

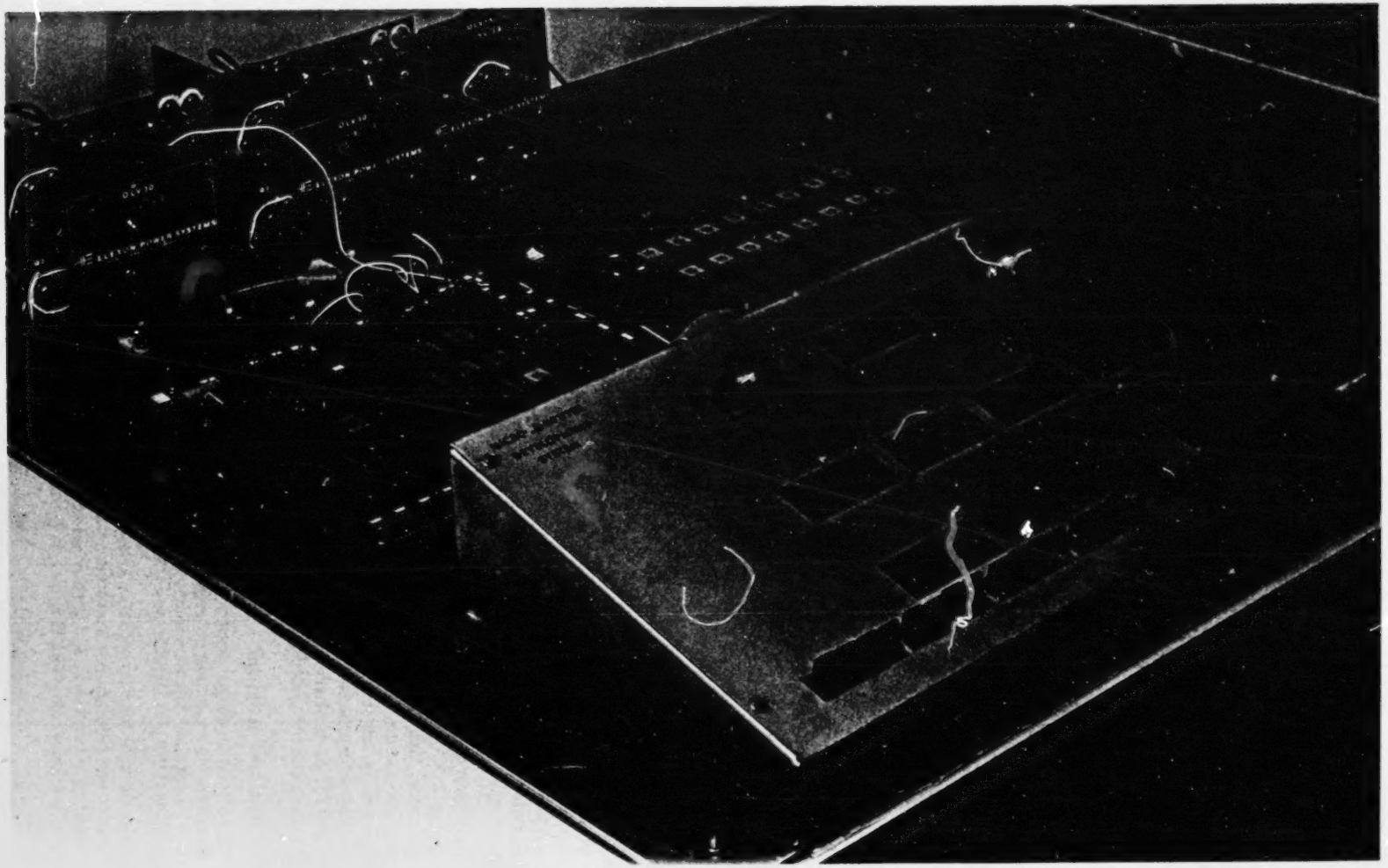

Figure 4. **Hardware development system.**

The basic concept is to write and simulate your microprocessor application program until it approximates the final product. Then the assembled program is loaded into the hardware system. The hardware emulates the proposed hardware design and executes the object code as it would be done with the final version.

At this point it is possible to test the micro-controlled hardware. If carefully designed, all is near completion. However, this is when the hardware may reveal some logic errors in the software or when the author's notions of the system design and implementation in his program fail to meet the actual physical requirements. However, using the hardware development system, with its many front panel controls, one can quickly and easily remove any remaining bugs from a program.

The development hardware includes three integrated circuit boards, a front panel with hexadecimal displays and switches, power supplies and a Plexiglas cover. The three large boards are functionally divided into a central processor board, memory board, and control panel board (Figure 5).

The central processor board includes the Motorola M6800 microprocessor, a crystal-controlled, one microsecond system clock, a one millisecond clock derived from this clock, a bootstrap loader, three peripheral interface adapters (PIAs), input and output buffering of the 16 bit address bus and 8 bit data bus, 768 bytes of random access memory (RAM) with a read/write protect switch, and a resident programmable read-only memory (PROM) programmer and socket for 4k and 8k Intel ultra-violet erasable PROMs.

The memory board has 3k bytes of random access memory arranged in three rows, 1k by 8 bits per row, with room on the board for another row. There is sufficient buffering of bus and data lines, and decoding logic for an additional 4k bytes of memory to be added by connecting another board. The memory board has a protect switch that disables the write function of the memory, thus making it into a psuedo ROM.

The third board is the control board. As compared to the MPU and memory boards, which are inherently clear as to the functions they perform, the control board should contain all the "bells and whistles" and development aids necessary to facilitate development and debugging of hardware and software. Initially, it is not always clear what these features are and which are best to have, but the necessity for a powerful and flexible control board with panel will manifest itself during the development process.

With this brief overview of the development hardware, a chronological description of a development process will show how the system is used, some of its authoring and debugging features, and the power and efficiency with which one person can develop an elaborate, microcontrolled, interactive, hardware/software system.

The development cycle is a continuous process of authoring software, editing, assembling, simulating, and emulating in hardware, with implied debugging throughout. This sequence is repeated in one form or another until a final product is produced. A program is interactively written in PIMPL at an AIS terminal. When assembled, the program is stored in the disk system for transfer to the hardware emulator. To serially transfer the object code, an AIS CAMIL (Computer Assisted/Managed Instructional Language) program called TRANSPORT is run at the AIS terminal.

TRANSPORT will automatically scan the PIMPL program data for declared ROM starting at address 0000 and scanning to the full addressing width of 65,535. For each contiguous segment of code, normally divided into 256 byte pages, TRANSPORT will serially output the high and low byte starting address and the number of bytes to be transmitted, followed by the actual object code data. The CAMIL language has a unique feature incorporated into its compiler and interface software for outputting data through an external output jack at the rear of the AIS terminal. The CAMIL software command, when executed, serially outputs a 20 bit word, with a timing pulse for each bit, and a sync signal for the end of each transmitted word. The output data rate is sixty, 20 bit words per second with the last 8 bits of each word serving as the actual data byte. Thus, in one minute about 3,600 words of PIMPL program may be transferred to the hardware emulator.

While TRANSPORT is running, the AIS terminal displays each page number and page length being sent. When the serial output of all PIMPL object code is complete, a message will appear at the terminal's screen stating that the output operation is finished. The operator then sets the memory protect switch located on the memory board of the hardware emulator. This has the effect of making the random access memory array of the hardware emulator appear as if it were now read only memory and nondestructive.

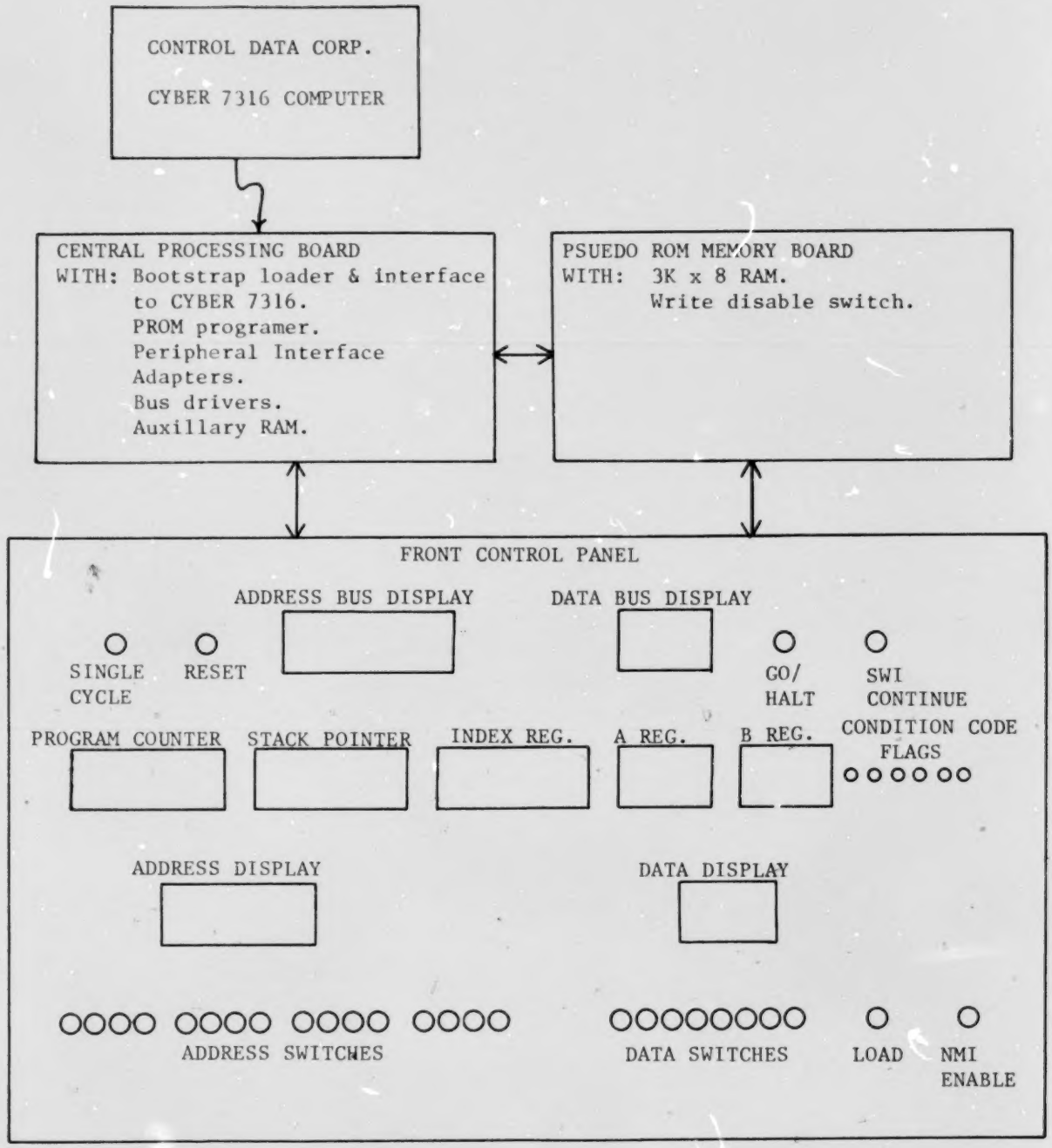

Figure 5. **Hardware development system.**

In order to receive data from the AIS computer, the hardware emulator uses a program called a bootstrap loader. One of the three PIAs on the central processing board is used for this interface. Coaxial lines are used between the terminal and emulator with SN75451 line drivers and SN7414 Schmitt receivers. Because this technique is good for frequencies to 20 MHz over distances of 100 feet, no parity is sent and no error detection or correction is used. As it turns out, several dozen transfers, with 24 thousand bits per transfer, failed to produce one error during emulation.

The bootstrap loader, once loaded and turned on, causes the emulator MPU to sit in a wait mode until interrupted by an external interrupt from the PIA. As the data is serially received at the emulator, the receipt of each full word causes a serial to parallel 8 bit shift into the PIA and an MPU interrupt. The MPU stores the byte in the next appropriate, sequential location of memory and then waits for another byte. Initially, at the beginning of each transfer the loader expects the high and low address and length to be sent, followed by the data. The MPU is fast enough to service each interrupt and byte sent to it through the PIA, and yet stay well ahead of the data transfer rates.

The bootstrap loader may be hand toggled into RAM by using the address and data switches on the front control panel, or a ROM version residing on the MPU board may be used after its starting vector is toggled in at the highest two addresses of memory.

The entire process of recompiling a large PIMPL program, transferring it by using TRANSPORT and the bootstrap loader, and running a new version of the program in the emulator, takes on the average between three and five minutes. A three to five minute turn-around time becomes an invaluable development aid. The programmer's output increases manyfold and complex microcomputerized systems are not only realizable, but with minimal time and effort.

Features and controls of the hardware development terminal or emulator that facilitate user development of software and hardware include:

1. complete microcomputer emulation
2. resident bootstrap loader for swapping programs from the CDC Cyber computer
3. psuedo ROM via RAM read/write protect switch
4. reset, halt and run switches
5. single cycle with hexadecimal display for address and data bus
6. software controlled breakpoints with display of all internal MPU registers
7. address controlled breakpoints with display of all internal MPU registers
8. breakpoint continue/run switch from trap
9. 16 address and 8 data switches with read/load control
10. peripheral interface adapters for emulation of prototype hardware
11. automatic PROM programmer with socket

The thought process by which an author debugs his program is unique unto himself. The intent of the type of controls provided, is to give him enough flexibility and capability to perform any type of logical debugging he desires. Normally, an assembled program listing (showing the address of each line of program code) is used in conjunction with the emulator. The program logic flow is followed on the listing while emulating. When something goes astray, the control panel switches and display "windows" enable the designer to "look inside" his program to learn what is specifically happening during execution.

After a program is first loaded into the emulator by the bootstrap, the RESET switch is pressed, followed by the RUN switch. Then while running, the MPU may be halted at any time by flipping the RUN switch to the HALT position. Once halted, control of the bus is given to the front control panel. The address display (4 hexadecimal digits) will show the last address on the address bus prior to halting the MPU. Likewise, the data display (2 hexadecimal digits) will display the last data to appear on the data bus prior to halting. If the MPU is not halted, the address and data are still displayed, but change at the processing speed of the MPU, causing the display to appear as a blur to the human eye.

When in the halt mode, each successive depression of the single-cycle switch will permit the MPU to execute the next program command. This action also updates the address and data displays. With this

feature, the author may "walk" through his program one step at a time while comparing the address and data displayed with the logic of his program listing.

If additional information is necessary for debugging, the internal registers and condition codes of the MPU may be displayed on the front panel. Display of this information is initiated in two ways. The author may insert the PIMPL command SWI (software interrupt) in his program at the point the display of information is required or he may toggle the appropriate program address on the front panel and enable the NMI (nonmaskable interrupt) switch (also on the front panel).

Both the SWI and NMI cause the MPU to display the value of its registers at the point in the program where the command is encountered, and then halt the MPU and emulation process. Information displayed includes the program counter, stack pointer, index register, A register, B register and condition code flags for carry, overflow, zero, negative, interrupt, and half carry. This information may be correlated with the program listing for possible bugs. When complete, the MPU may be restarted by hitting the SWI/NMI continue switch on the front panel.

This information is normally not available to designers because of no access provisions on the MPU integrated circuit. However, the SWI/NMI feature of the emulator causes the MPU to jump into a "priority display and halt" mode. When a software or nonmaskable hardware interrupt is encountered by the MPU, it places its internal contents in the memory stack while processing the interrupt. The emulator copies the information stored in the stack, displays it on the front panel in a hexadecimal format, restores the MPU from the stack, and then halts the MPU. This entire process is transparent to the program running in the hardware emulator, but provides a powerful aid to the developer.

When the hardware and software development process using the simulator and emulator is complete, the designer should have a smoothly running system that is a version of the machine he eventually hopes to build. All the logic design for the microprocessing unit, input/output interface, external signals and software should be complete. All that remains for the designer is to reconfigure his hardware as a stand-alone system (i.e., no longer using the emulator, but instead, the actual integrated circuits) and to transform his assembled software code from the psuedo read only memory of the memory board to ROM or PROMs.

Since the emulator served as a hardware prototype, the reconfiguration to a stand-alone microcomputerized system from the emulator is a straightforward process. The electronic schematic diagrams and circuits used during emulation are the same for the final product.

Programming PROMs may be done quickly and automatically using the hardware development terminal. The assembled code already resides in the psuedo ROM area of the memory board. The bootstrap loader in conjunction with the program, TRANSPORT, loads the PROM programmer program from the AIS computer into the hardware development terminal at a predesignated and reserved part of memory.

The program's listing of the memory map, provided at the end of each PIMPL assembly, is used to divide the object code into sequential sections of 512 or 1024 bytes of code. These sections provide the code for each PROM to be programmed.

To program a PROM, an ultraviolet erasable 4k or 8k PROM (Intel's 2704 or 2708) is placed in the socket located on the MPU board. The 16 bit starting address (high and low order byte) for the section of code to be programmed is toggled into memory, at location 0000 and 0001 using the front panel controls. The PROM programmer program is RESET and RUN. A small red light next to the PROM socket will be lit during the programming cycle. When the cycle is complete, approximately 2 1/2 minutes for a 4k PROM, the light turns off, the PROM may be removed, and the next PROM may be programmed using the same process.

It becomes a simple process to reprogram the PROMs for a new version of software. To do this, Intel PROMs are exposed one inch from an ultraviolet light source for 20 to 30 minutes. This erases the old bit pattern stored in the PROM, and readies the PROMs for a new programming cycle.

IV. DESCRIPTION OF THE STUDENT MICROTERMINAL

The following section will give a description of the final version of the student microterminal explaining its physical layout, resident controlling program and microcomputer electronics.

Refer to Figure 1 for an illustration of the microterminal. The box shown measures approximately 10 by 5 by 3 inches, yet contains the entire electronics and memory for a stand-alone computer that has five testing strategies and 900 multiple-choice answer patterns.

Messages used in a dialog with the student are conveyed by means of lighting a small red light adjacent to each printed message on the front panel. The current version has fourteen small lights for messages. Six of these are used for messages with the student while administering a test, two more are for yes/no feedback, four are used when extracting data collected while in the instructor mode, and the last two lights are unused. It is possible to add more lights or to have more than one light on at a time, if multiple messages are desired simultaneously.

The display located in the middle has four light emitting diode (LED), hexadecimal displays which display the numbers between 0 and 9 and the characters A through E for multiple-choice response feedback. Larger numbers, such as a nine digit social security number, are displayed by stepping the numbers across the four-digit display from right to left as they are entered.

The keyboard's sixteen keys include the numbers 0 through 9, CLEAR, SEND, and four blanks. The keys numbered between 1 and 5 also have the letters A through E printed on them for multiple-choice responses. Several different manufactured keyboards were tested until one was found which had a positive tactile feel and whose key caps could be altered for messages appropriate to our testing scenario.

The prototype box is made from one-eighth inch and three-sixteenth inch white opaque Plexiglas. This has proven to be an easy material to work with and has withstood a lot of rough handling.

The PIMPL program that controls the box resides in six 512 by 8 bit, ultra-violet erasable PROMs, four of which are for the program and two more for test answer patterns. If the testing pattern is ever compromised, the two PROMs holding the answer pattern may be easily exchanged for two new ones without affecting or altering the other four PROMs.

The PIMPL program is sequentially arranged in the order of declarations, answer arrays, procedures, program body, and initialization of hardware vector pointers. Logically the program is executed by first initializing variables, then by receiving the user's social security number and test booklet number, followed by administration of the test, and concluding with the calculation and display of the final test score.

The type of test strategy to be used, the answer pattern for the test administered and the length of the test are all derived from the five-digit test booklet number that the student enters.

Extensive software was developed to prevent the student from entering erroneous data, such as a nonexistent or invalid booklet number, and to prevent him from accessing proprietary information. In every case, the microterminal box has not been difficult to use by students attempting to use it for the first time. The lights and messages in a natural progression through a testing strategy, seem to be well human factored and not confusing to new users.

Because of the nature of the PIMPL language and the extensive use of procedures in developing the microterminal program, the actual amount of code generated is quite compact. All together, 29 separate procedures are used, with nesting of called procedures several levels deep. This approach makes software development that controls and replaces hardware much easier and more logical to the developer. The actual body of the program is relatively short compared to the rest of the program listing, but it visibly contains the entire outline and structure of the program.

When tasks are required, such as receiving 9 keyed inputs from the student for his social security number followed by a SEND key, a procedure called ACCEPT is simply called with the variable, LENGTH, initialized to nine prior to the call. ACCEPT (in turn) uses other procedures, which in turn call other segments of code. In this way, code is used very efficiently.

It is important to note that when designing a hardware/software system, simple trade-off studies are necessary to determine if a control signal, electrical pulse or some function is more efficiently implemented in software or in hardware. With the current availability of large read only memories at low prices combined

with the power, speed and flexibility of the microprocessor it becomes very cost-effective to implement as much of the control hardware in software (also referred to as firmware) as possible.

This approach was used in the design of the microterminal. A good example of this is in how the keyboard is handled. Scanning of the keyboard for depressed keys, encoding of the keys, debounce of keys during depression, and key lockout if more than one key is pressed at the same time is all handled by the microprocessor and firmware.

For the prototype microterminal, five testing strategies were implemented and reside in the "box." The first strategy is merely a linear progression through a test, with no correct response feedback. The second strategy is a linear progression with yes/no response feedback. The third strategy is similar to the second, except that when a test item is missed the student must remain at that item until he gets it correct. The fourth strategy is the same as the third except that at the end of the test, the student loops back around and continues taking each question he missed, until he has answered them correctly.

The fifth strategy implements a form of adaptive testing, called flexilevel. The intention of adaptive testing is to determine a student's comprehension level of the subject matter by using an algorithm that asks as few questions as necessary during the test, but which maintains as high a correlation as possible to the test score that would have been derived had all the test questions been asked.

In a flexilevel adaptive test, the questions are ranked from the first to the last item according to increasing degrees of difficulty. In the strategy implemented, the first question administered is the middle question of the test. Each successive question asked is based on whether the preceding question was answered correctly or incorrectly, a higher numbered question for correct and a lower numbered question for incorrect. In this manner the test continues until the student finishes at the last question or the first numbered question of the test. The fifth strategy uses this technique, using any variable length test and always starting with the numerically middle question.

At the present time, a hardware interface is being developed which will enable the microterminal to swap its gathered student test data into the central computer-managed system at the completion of a test.

For the present time, until the central interface is complete, information may be manually retrieved from the terminal while in an instructor mode. To log on as an instructor a special sequence of keys is pressed. Once this is done a red light will appear next to the message that says "Instructor Mode."

To retrieve information while in instructor mode, a key corresponding to the same number of a light with appropriate message is depressed. For instance, if when in instructor mode a 1 is depressed, the nine digit social security number may be stepped across the display. A 2 will display the booklet number. A 4 will display the student's score, while a 5 will display the elapsed time taken during the test, as expressed in hours and minutes. Key 6 depressed (used in conjunction with the SEND key) will display each missed question and all the incorrect responses given for that question. For each of these instructor functions, appropriate lights and messages are used to facilitate the retrieval process. The microterminal is reset for a new student by depressing the blank key under the CLEAR key while in the instructor mode.

It was mentioned previously that the elapsed time during a test was measured. This capability is performed by an interesting combination of hardware and software. The microprocessor uses a one microsecond crystal-controlled system clock. From this frequency, a millisecond clock is derived in hardware by using three successive decade counters. This millisecond clock interrupts the microprocessor each one-thousand of a second. Slower clock periods are derived from this interrupt. Each interrupt increments a software millisecond counter which in turn affects one-tenth second, second, minute and hour software variable clocks.

Not only is this timing mechanism used for measuring test durations, but it serves many other purposes, such as for short time delays in key debounce routines, strobe pulses for latching data to the microterminal displays, and for timed pauses when displaying messages such as the yes/no test item feedback.

The microterminal hardware consists of a Motorola M6800 microprocessing unit, a 128 x 8 bit random-access memory chip, six 4096 bit programmable read only memories, a 1 megahertz clock, one peripheral interface adapter, fourteen discrete light emitting diodes, four hexadecimal displays, a sixteen key keyboard, and a few discrete transistor-transistor locic (TTL) integrated circuits. Except for the keyboard, all these parts fit on a 4.5- by 6.0-inch board that mounts against the underside of the top panel

of the microterminal box. By maintaining a compatible family of microcomputer integrated circuits, such as the Motorola 6800 family, and by performing many of the hardware control functions in software the total parts count was kept low thus enabling the building of the entire microcomputer with appropriate student interactive displays on one small board. Refer to Figure 6 for a functional block diagram of the complete microterminal.

V. CONCLUSION

The stand-alone testing terminal, described in this report, is the tip of an iceberg which could greatly change the manner in which testing is performed in the armed services. Specific systems may be relatively easily developed which emphasize testing technique, test item security, learning during testing, or automated entry of test results for item or response analysis by a larger system. The greatest difficulty will be obtaining assembled devices such as these in small quantity without losing the low-cost potential of the imbedded microcomputer, and in attaching families of these devices to current inappropriately designed central systems. If solutions to these problems can be found, the full potential of reliable automated, self-paced learning centers may be realized.

☆U.S. GOVERNMENT PRINTING OFFICE: 1977-771-057/29